Heuristic Learning and Search Without Exhaustive Action Enumeration

Misagh Soltani, Forest Agostinelli

Department of Computer Science and Engineering
University of South Carolina, USA
msoltani@email.sc.edu, foresta@cse.sc.edu

Abstract

The combination of deep reinforcement learning and heuristic search has proven capable of solving pathfinding problems with relatively little domain-specific assumptions. However, these methods do not currently extend to domains whose action spaces are too large to enumerate exhaustively, such as continuous action spaces. In this work, we address this problem by introducing a policy-guided method for heuristic search over such action spaces. We use a stochastic policy function that samples a limited number of actions for each state. Sampled approximate dynamic programming is then used to learn a heuristic function that maps states and goals to their estimated cost-to-go. This action sampling process is also used when performing heuristic search with the learned heuristic function. We evaluate our approach on the continuous control problem of Ant Maze, assuming access to the transition function. Although we limit the stochastic policy to a simple random uniform distribution, our results also show that heuristic search methods such as beam search and batch weighted A* search can find significantly shorter paths compared to behaving greedily with respect to the learned heuristic function. This shows the benefit of combining deep reinforcement learning with heuristic search, even when action enumeration is not possible.

Introduction

Deep reinforcement learning (Sutton and Barto 2018) has been used to learn domain-specific heuristic functions that, when combined with heuristic search (Hart, Nilsson, and Raphael 1968), create solvers for pathfinding problems whose efficacy rivals handcrafted solvers and domain-independent solvers based on formal domain definitions (Agostinelli et al. 2019; Zhang et al. 2020; Bao and Hartnett 2024; Agostinelli, Panta, and Khandelwal 2024; Panta et al. 2024; Turner, Agostinelli, and Fu 2025). However, such methods have been restricted to finite, discrete, action spaces due to both the reinforcement learning and heuristic search algorithms assuming the ability to enumerate all possible actions. This makes applying such algorithms to very large discrete action spaces, such as those found in multi-agent pathfinding, impractical and to infinite action spaces, such as those found in robotics with continuous actions, impossible.

To overcome this, we introduce a method of learning a heuristic function and performing heuristic search by using a stochastic policy function to sample actions. We evaluate our approach on the Ant Maze (Fu et al. 2020) continuous control pathfinding task. We show that the combination of this approach with heuristic search results in lower path costs compared to behaving greedily with respect to the learned heuristic function. As a result, we show that the benefits of the combination of deep reinforcement learning and heuristic search can be extended to domains where actions cannot be exhaustively enumerated.

Related Work

Reinforcement Learning Without Action Enumeration

Policy gradient methods have been used to map states to probability distributions over actions and take gradient steps to increase the probability of beneficial actions (Williams 1992). In enumerable action spaces of a fixed size, the probability distribution can be obtained by normalizing the output of the policy function. However, in continuous action spaces, this is typically accomplished by first assuming the probability distribution comes from a certain class and using the output of the policy function to parameterize that distribution. For example, assuming the distribution class is a spherical normal distribution and parameterizing the mean and standard deviation.

To remove the need to assume the class of probability distribution, deterministic policy gradient algorithms can be used to deterministically map states to actions and take gradient steps to maximize an action-value function (Silver et al. 2014). However, exploration still requires manually constructing a distribution from which noise is sampled. On the other hand, policies that are agnostic to the class of probability distribution can be represented by a conditional generative model that conditions on the state to sample actions. Such architectures, such as conditional variational autoencoders (Kingma and Welling 2013; Sohn, Lee, and Yan 2015), normalizing flows (Dinh, Krueger, and Bengio 2014; Rezende and Mohamed 2015), transformers (Vaswani et al. 2017), and diffusion models (Ho, Jain, and Abbeel 2020), make no assumptions on the distribution of the actions besides those implicitly encoded by the architecture of the

generative model. Tessler, Tennenholtz, and Mannor (2019) demonstrate that restricting the class of distributions can result in sub-optimal solutions and propose using quantile regression to overcome this. Ward, Smofsky, and Bose (2019) and Mazoure et al. (2020) combine normalizing flows with soft actor-critic (Haarnoja et al. 2018) to create policies capable of learning multi-modal distributions. Wang, Hunt, and Zhou (2023) and Mark et al. (2024) distill a policy obtained from an action-value to policies represented with diffusion models and transformers.

Heuristic Search Without Action Enumeration

A direct way to handle large branching factors in best-first search is to make node expansion partial. Yoshizumi, Miura, and Ishida (2000) propose A* with Partial Expansion, which generates all children of a node but stores only those with the same f-cost and re-inserts the parent with the f-cost of the next best unstored child. Felner et al. (2012) extend this idea with Enhanced Partial Expansion A*, which uses domain- and heuristic-specific operator selection functions to generate only children with a desired f-cost. Recent work on planning with infinite domain control variables explicitly use sampling. This is done by partially expanding states by sampling successors and re-expanding promising states to obtain additional samples (Aso-Mollar et al. 2025b,a). These methods reduce the number of generated or stored successors while keeping the structure of best-first search. However, they focus on controlling expansion for a given heuristic or sampler rather than on learning a goal-conditioned heuristic function when all actions cannot be enumerated.

Larger finite action spaces can arise in domains where a single action represents a joint action taken by several agents. In multi-agent pathfinding (MAPF) problems, operator decomposition and independence detection help to reduce the branching factor by decomposing the joint actions either into individual choices or by finding and solving subproblems for groups of agents that do not interact with each other (Standley 2010). Conflict-based search uses a different factorization by searching over collision constraints at the higher level and solves single-agent pathfinding problems at the lower level (Sharon et al. 2015). These methods circumvent the need to explicitly consider the entire action space by exploiting known product structure in the problem, and, therefore, depend on structure that may not available in a more general setting.

Another line of work avoids enumeration of continuous actions by sampling actions instead of expansion. In task and motion planning, PDDLStream (Garrett, Lozano-Pérez, and Kaelbling 2020) uses constraints over continuous variables via black box streams, transforming planning into a sequence of a finite set of PDDL problems. Learned samplers exploit previous planning experience before to bias these calls. Kim, Kaelbling, and Lozano-Pérez (2018) learn an action sampler for use in continuous state-action search and Kim, Kaelbling, and Lozano-Pérez (2019) learn a policy for choosing continuous action parameters through search. Monte Carlo tree search (MCTS)-based approaches also utilize a similar approach by constructing and refining a set of candidate actions using kernel regression, value gradient refinement, optimistic sampling, or Bayesian optimization (Yee et al. 2016; Lee et al. 2020; Kim et al. 2020; Mern et al. 2021). In these methods, sampling is used to make search possible in continuous or very large action spaces. However, their primary goal still lies in choosing action parameters for a planner or estimating values inside online planners rather than on learning a heuristic function that can be used by a search algorithm without requiring action enumeration.

Preliminaries

Pathfinding

We seek to solve pathfinding problems in very large or continuous action spaces. A pathfinding domain is defined as a weighted directed graph (Pohl 1970), where nodes represent states, edges represent actions that transition between states, and edge weights represent transition costs, where all transition costs are greater than 0. Note that a node can have an infinite number of edges, such as in the case where the action space is continuous. The transition function is denoted T and $s' = T(s, a)$ if and only if there exists an edge between s and s' that corresponds to action a. The transition cost function is denoted c and $c(s, a)$ is the transition cost when taking action a in state s. The set of all possible actions is denoted $\mathcal{A}$.

A pathfinding problem is defined by a pathfinding domain, a start state, and a goal, where a goal represents a set of goal states. A path is a sequence of actions and the path cost of a path is the sum of transition costs. A solution to a pathfinding problem is a path that transforms the start state into a goal state with a preference for lower path costs. A path that has the lowest possible path cost amongst all possible solutions to a given pathfinding problem is referred to as a shortest path or an optimal path and its cost is referred to as the cost-to-go.

DeepCubeA

DeepCubeA learns a heuristic function for a goal-conditioned pathfinding problem by combining approximate value iteration with heuristic search (McAleer et al. 2019; Agostinelli et al. 2019). The heuristic function, $h_\theta(s, g)$, is trained to predict the cost-to-go from a state s to a goal g. The target for each state is computed by the Bellman backup in Equation 1, where h_{θ^-} denotes a target network that is periodically updated to θ (Mnih et al. 2015). Note that Equation 1 assumes the ability to minimize over all possible actions. The neural network parameters are then updated by minimizing the error between $h_\theta(s, g)$ and $h'(s, g)$.

$$h'(s,g) = \begin{cases} 0, & \text{if } s \in g, \\ \min\limits_{a \in \mathcal{A}} c(s,a) + h_{\theta^-}(T(s,a), g), & \text{otherwise.} \end{cases} \tag{1}$$

At test time, DeepCubeA uses the learned heuristic with batch weighted A* search (BWAS) (Pohl 1970). BWAS uses a weight to decrease the influence of the path cost to potentially find solutions faster and batches node expansion to take advantage of parallelism provided by graphics processing units for deep neural networks (DNNs). Similar to value

iteration, BWAS assumes nodes are expanded using all possible actions. This assumption prevents algorithms such as DeepCubeA from being applied to action spaces where exhaustive enumeration is impractical or impossible

Methods

Heuristic Function Learning

When learning the heuristic function, we assume the existence of a policy from which actions can be sampled given a state and goal, $a \sim \pi(\cdot|s, g)$). We then build on sampled dynamic programming (Atkeson 2007) to learn a heuristic function that maps states and goals to the expected sum of transition costs when acting according to π, $h_\pi(s, g) = \mathbb{E}_{a\sim\pi}[\sum_{k=0}^{\infty} c(s_{t+k}, a_{t+k})|S_t = s, G = g]$, where $c(s_i, a_i) = 0$ if $s_j \in g$ for some $j \leq i$. The heuristic function is represented with a deep neural network (DNN) with parameters θ. Given state and goal pairs, the DNN is trained to minimize the loss function in Equation 2 with stochastic gradient descent (Rumelhart, Hinton, and Williams 1986). Here, h'_π is similar to the value iteration update in Equation 1, except $\mathcal{A}$ is replaced by $\alpha_{s,g} = [a_0, ..., a_{A-1}]$, $a_i \sim \pi(\cdot|s, g)$, and A is the number of actions to sample from π.

$$L(\theta) = \frac{1}{N} \sum_{i}^{N} (h'_\pi(s_i, g_i) - h_\theta(s_i, g_i))^2 \tag{2}$$

To obtain states and goals for training, we assume the ability to sample problem instances. Given a problem instance, an attempt is made to reach the goal following a stochastic policy derived from the heuristic function, where each action is sampled according to a Boltzmann distribution, as shown in Equation 3, where τ is the temperature. The heuristic function is given a maximum of K steps to find a path to the goal. Each state generated from attempting to reach the goal is added to the training set and is paired with its corresponding goal. In the case where a path is not found, we use hindsight experience replay (HER) (Andrychowicz et al. 2017) by sampling a new goal g' from the terminal state s_{K-1}, where $s_{K-1} \in g'$. We then replace the original goal with g' for training. This approach has been shown to aide in reaching progressively more difficult goals as a function of training time.

$$p_{s,g,a} = \frac{e^{-h_\theta(T(s,a),g)/\tau}}{\sum_{a' \in \alpha_{s,g}} e^{-h_\theta(T(s,a'),g)/\tau}}, \tag{3}$$

Heuristic Search

The heuristic search methods we use are batch weighted A* search (BWAS) (Pohl 1970; Agostinelli et al. 2019; Li et al. 2022) and beam search. When expanding a node, we use sampled edges $\alpha_{s,g}$. This ensures that any path found is a valid path as π only samples edges from the original weighted directed graph. However, this approach no longer guarantees optimality (when given an admissible heuristic function (Hart, Nilsson, and Raphael 1968)) or completeness. Setting A, the number of actions to sample, is a tradeoff between solution quality and speed. An A to low increases the risk of omitting useful actions, thus resulting in higher path costs and fewer problem instances solved. An A too high makes heuristic search time impractical.

Experiments

Domain

We evaluate our approach on the Ant Maze task (Fu et al. 2020) ("AntMaze_UMaze-v5", specifically) implemented in the OpenAI Gym (Brockman et al. 2016). We assume access to the underlying simulator to compute the transition function and set all transition costs to 1. Ant Maze consists of a quadruped "ant" robot, a goal location, and a maze layout. States include the coordinates of the robot's torso, angles between the torso and its links, angles between the joints, angular velocities, as well as contact forces information for each link. This is represented as a 105 dimensional vector. The goal is an x,y coordinate in the maze and the goal is reached if and only if the Euclidean distance between the robot's torso and the goal is less than 0.5 meters. The maze layout is a fixed "U" shape. An action is a set of 8 torques applied to each joint, where the torque is a continuous value between -1 and 1. Therefore, an action is represented as 8 dimensional vector.

Training

The policy we use to obtain actions is a random uniform policy. Therefore, each element in the 8 dimensional vector representing the sampled action is drawn from a random uniform distribution between -1 and 1. The number of actions sampled is set to $A = 10$ for training and heuristic search. For HER, we use the robot's current torso location to generate a goal from a state. Problem instances are sampled from the default Ant Maze environment, which randomly places the robot and goal location at reachable points in the maze. We set $\tau = 1$ for attempting to reach goals with the stochastic policy and set $K = 100$ to be the maximum number of steps.

States and goals are represented to the heuristic function as the 105 dimensional vector describing the state of the robot along with a 2 dimensional vector representing the goal and another 2 dimensional vector representing the difference between the current coordinates of the robot's torso and the goal coordinates. The input to the heuristic function is fed to a linear layer of size 1,000, which is fed to a residual neural network (He et al. 2016) with four blocks and two hidden layers of size 1,000 per block. This is fed to a linear layer of size 1 which represents the estimated cost-to-go. We use batch normalization (Ioffe and Szegedy 2015) and rectified linear units (Glorot, Bordes, and Bengio 2011) in all residual layers. The heuristic function is trained for 100,000 iterations with a batch size of 1,000. Optimization is performed with ADAM (Kingma and Ba 2014) with a learning rate of 0.001 and a decay rate of 0.9999993. The percentage of states solved with the stochastic policy used to obtain data as a function of training iteration is shown in Figure 1. The figure shows that the heuristic function initially solves close to 0% of problem instances and steadily improves as the training iteration increases.

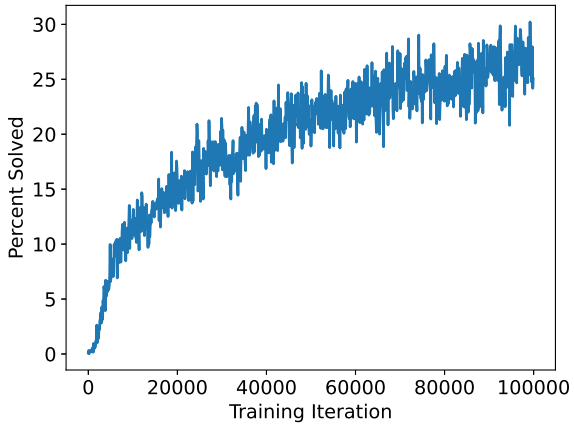


Figure 1: The percentage of states solved with a stochastic policy derived from the heuristic function as a function of training iteration.

Heuristic Search

After the heuristic function is trained, we use it to solve 100 randomly generated problem instances with the following algorithms: a greedy policy, beam search with a beam size of 10, and BWAS with combinations of a batch size of 1 and 10 and a weight on the path cost of 0.0, 0.5, and 1.0. The greedy policy samples A actions at each state and takes the action that corresponds to the action whose resulting state has the lowest estimated cost-to-go according to h_θ. We also compare to solving problem instances with just a random policy and no trained heuristic function. Each algorithm is allowed 2 minutes per problem instance to find a solution.

Results for each of the pathfinding algorithms ran on the test problem instances is shown in Table 1. The table shows that all algorithms that use a heuristic function perform significantly better than the purely random policy, especially in terms of path cost and solution time. For example, beam search solves a higher percentage of problems while being one order of magnitude faster and having an average path cost that is three orders of magnitude smaller than the random policy. This shows that learning a heuristic function with actions sampled from just a random policy can lead to significantly better solutions. This is in line with results in reinforcement learning literature that show improvements when performing policy evaluation on a random policy can lead to significant improvements (Sutton and Barto 2018).

Amongst the algorithms that use a heuristic function, the greedy policy solves the highest percentage, with beam search and certain BWAS settings being very close to it while also having significantly lower path cost[1]. To better understand the difference in path cost, Figure 2 shows the path cost for each problem instance amongst all pathfinding algorithms. The figure shows that larger batch sizes and larger weights on the path cost tends to result in shorter paths, when they do solve problems.

[1]Since the majority of computation time was consumed by the simulator, the advantage of parallelism found in beam search and BWAS due to speeding up heuristic computation with GPUs has yet to be realized.

Qualitative Evaluation

To better understand how heuristic search obtained more efficient solutions compared to the greedy policy, we visualize the trajectories for specific problem instances where there is a large difference in path cost. Figure 3 shows an instance where the greedy policy takes 664 steps because the robot falls on its back and cannot get up. As a result, it resorts to wiggling itself to the goal. On the other hand, beam search solves the problem in 44 steps and the robot does not fall on its back. Figure 4 shows that the greedy policy, again, results in the robot falling on its back and wiggling itself to the goal. Beam search results in the robot landing on its back, but in a rolling motion so that it quickly recovers and jumps to the goal. A* search (BWAS with B=1 and W=1.0) also results in the robot rolling, but, after the roll, it jumps diagonally at the wall and lands at the goal. While the behavior of heuristic search is, indeed, more efficient when all transition costs are 1, this behavior can be dangerous in a real-world setting. One way to reduce this behavior could be to modify the transition cost function such that movements that result in the robot leaving the ground, landing on its back, or accelerating too quickly have a higher transition cost.

Future Work

Learned Policies

The method assumes access to a policy that can generate useful candidate actions. This policy can be hand-designed, trained separately, or obtained from another solver, but its quality directly affects the how accurately the learned heuristic function estimates the true cost-to-go and the optimality and completeness of heuristic search. While this work only examined a random policy, a natural extension is to learn the policy together with the heuristic. The heuristic can provide training signals for the policy by identifying sampled actions that lead to lower path costs, while the policy can improve the heuristic by sampling better successor states. The policy can take the form of a generative model, so that different actions can be sampled for learning and search without assumptions on action distribution, such as policies based on normalizing flows (Ward, Smofsky, and Bose 2019; Mazoure et al. 2020), diffusion models (Wang, Hunt, and Zhou 2023), and transformers (Mark et al. 2024). This would be an instance of policy evaluation and improvement, the foundation of reinforcement learning (Sutton and Barto 2018), applied to heuristic search with stochastic policies that can learn arbitrary probability distributions. Future work can also incorporate intrinsic motivation (Barto et al. 2004) to encourage policies to sample actions for which cost-to-go estimates are uncertain to add exploration during training.

World Models

The current formulation assumes that the transition function and transition cost function can be computed for any state

Solver	Cost	Nodes	Secs	Solved
Rand	4.45E+04	4.45E+04	46.36	49%
Greedy	2.25E+02	2.26E+03	3.61	77%
Beam (10B)	4.75E+01	4.76E+03	4.02	76%
BWAS (1B,0W)	6.26E+01	4.83E+03	7.5	73%
BWAS (10B,0W)	4.35E+01	7.30E+03	6.59	74%
BWAS (1B,0.5W)	4.35E+01	6.04E+03	6.11	67%
BWAS (10B,0.5W)	3.53E+01	5.91E+03	4.67	63%
BWAS (1B,1W)	3.59E+01	2.07E+04	21.94	50%
BWAS (10B,1W)	3.17E+01	2.58E+04	20.13	49%

Table 1: The performance of the pathfinding algorithms in terms of average path cost, number of nodes generated, solution time, and percentage solved. These numbers were only computed amongst the instances that were solved.

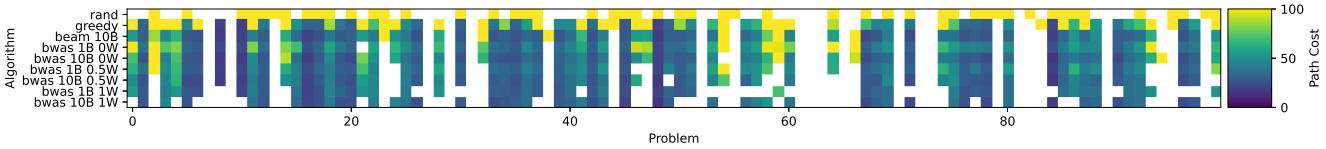
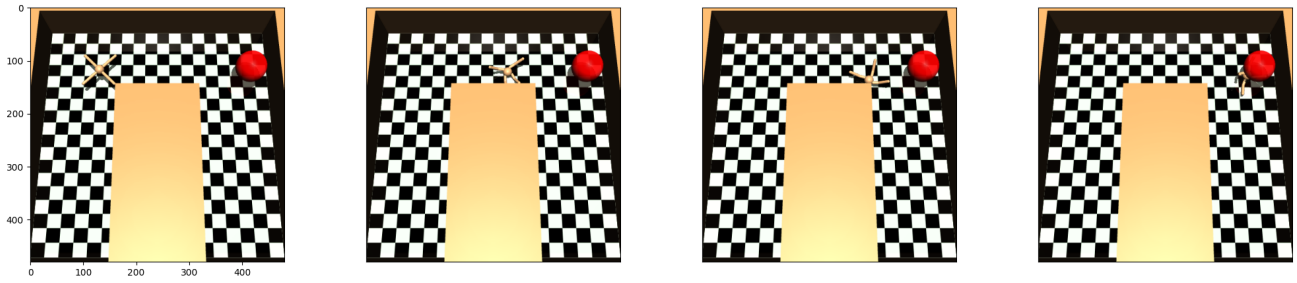


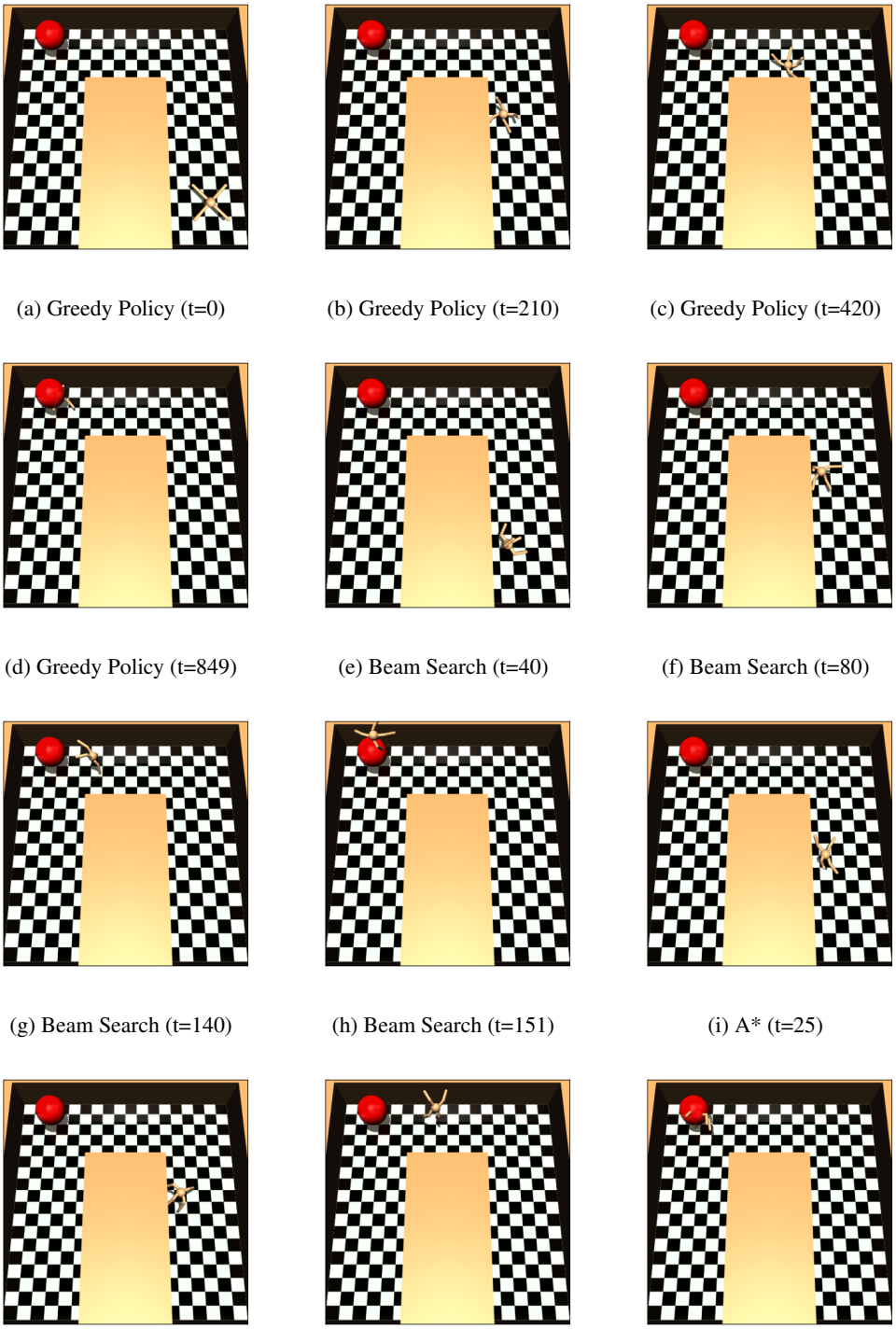
Figure 2: Path costs between 0 and (at least) 100 for all 100 problem instances. Of the instances that are solved, larger batch sizes and larger path cost weights tend to result in lower path costs.



(a) Greedy Policy (t=0) (b) Greedy Policy (t=220) (c) Greedy Policy (t=440) (d) Greedy Policy (t=664)

(e) Beam Search (t=0) (f) Beam Search (t=15) (g) Beam Search (t=30) (h) Beam Search (t=44)

Figure 3: The greedy policy results in the ant falling on its back and wiggling itself to the goal. Beam search finds a path such that the ant does not fall. The red sphere denotes the goal.



(a) Greedy Policy (t=0)

(b) Greedy Policy (t=210)

(c) Greedy Policy (t=420)

(d) Greedy Policy (t=849)

(e) Beam Search (t=40)

(f) Beam Search (t=80)

(g) Beam Search (t=140)

(h) Beam Search (t=151)

(i) A* (t=25)

(j) A* (t=30)

(k) A* (t=57)

(l) A* (t=70)

Figure 4: The greedy policy results in the ant falling on its back and wiggling itself to the goal. This also happens during beam search, but the robot quickly rolls and reaches the goal. For A* search, the robot tumbles its way towards the goal, leaps off the ground, bounces off the wall, and lands at the goal.

and action. In many applications, such as those in robotics and physical simulation, computing transitions may be expensive or the true transition function may not be known. World models (Sutton 1991; Schmidhuber 2015) have been used to overcome this by using DNNs to approximate the transition function and transition cost function (Oh et al. 2015; Asai et al. 2022; Tian et al. 2021). It has been observed that world models that operate in a discrete latent space tend to accumulate significantly less error compared to those that operate in a continuous space and can also be used to solve problems with heuristic search (Agostinelli and Soltani 2024). However, for some real-world problems, it may be the case that any given world model will accumulate a significant amount of error after a certain number of timesteps. Therefore, these approaches may have to be combined with re-planning when the states on the computed path deviates significantly from states on the observed path.

Conclusion

We present a method for heuristic learning and search that does not require exhaustive action enumeration. This method can be applied to problems where exhaustive action enumeration is impractical or impossible, such as continuous action spaces. We focused on using a random policy to sample actions for heuristic learning and search and showed that employing heuristic search plays a significant role in obtaining more efficient solutions. Future work will focus on iterating between policy evaluation and improvement to obtain iteratively better heuristic functions and policies. Future work will also combine this approach with world models to generalize this approach to real-world robotics problems without simulators and to potentially speed up transition function computation.

Acknowledgments

This material is based upon work supported by the National Science Foundation under Award No. 2426622. The authors gratefully acknowledge the computational resources provided by the Theia high performance computing cluster at the University of South Carolina which is supported by National Science Foundation Grant No. 2320292. We also acknowledge the technical assistance and resources provided by Research Computing at the University of South Carolina (RRID:SCR_027488).

References

Agostinelli, F.; McAleer, S.; Shmakov, A.; and Baldi, P. 2019. Solving the Rubik's cube with deep reinforcement learning and search. *Nature Machine Intelligence*, 1(8): 356–363.

Agostinelli, F.; Panta, R.; and Khandelwal, V. 2024. Specifying goals to deep neural networks with answer set programming. In *34th International Conference on Automated Planning and Scheduling*.

Agostinelli, F.; and Soltani, M. 2024. Learning discrete world models for heuristic search. In *Reinforcement Learning Conference*.

Andrychowicz, M.; Wolski, F.; Ray, A.; Schneider, J.; Fong, R.; Welinder, P.; McGrew, B.; Tobin, J.; Abbeel, O. P.; and Zaremba, W. 2017. Hindsight experience replay. In *Advances in Neural Information Processing Systems*, 5048–5058.

Asai, M.; Kajino, H.; Fukunaga, A.; and Muise, C. 2022. Classical planning in deep latent space. *Journal of Artificial Intelligence Research*, 74: 1599–1686.

Aso-Mollar, Á.; Aineto, D.; Scala, E.; and Onaindia, E. 2025a. Handling infinite domain parameters in planning through best-first search with delayed partial expansions. *arXiv preprint arXiv:2509.03953*.

Aso-Mollar, A.; Aineto, D.; Scala, E.; and Onaindia, E. 2025b. A sampling approach to planning with infinite domain control variables. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 35, 149–153.

Atkeson, C. G. 2007. Randomly sampling actions in dynamic programming. In *2007 IEEE International Symposium on Approximate Dynamic Programming and Reinforcement Learning*, 185–192. IEEE.

Bao, N.; and Hartnett, G. S. 2024. Twisty-puzzle-inspired approach to Clifford synthesis. *Physical Review A*, 109(3): 032409.

Barto, A. G.; Singh, S.; Chentanez, N.; et al. 2004. Intrinsically motivated learning of hierarchical collections of skills. In *Proceedings of the 3rd International Conference on Development and Learning*, volume 112, 19. Citeseer.

Brockman, G.; Cheung, V.; Pettersson, L.; Schneider, J.; Schulman, J.; Tang, J.; and Zaremba, W. 2016. Openai gym. *arXiv preprint arXiv:1606.01540*.

Dinh, L.; Krueger, D.; and Bengio, Y. 2014. Nice: Nonlinear independent components estimation. *arXiv preprint arXiv:1410.8516*.

Felner, A.; Goldenberg, M.; Sharon, G.; Stern, R.; Beja, T.; Sturtevant, N.; Schaeffer, J.; and Holte, R. 2012. Partial-expansion A* with selective node generation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 26.

Fu, J.; Kumar, A.; Nachum, O.; Tucker, G.; and Levine, S. 2020. D4rl: Datasets for deep data-driven reinforcement learning. *arXiv preprint arXiv:2004.07219*.

Garrett, C. R.; Lozano-Pérez, T.; and Kaelbling, L. P. 2020. Pddlstream: Integrating symbolic planners and blackbox samplers via optimistic adaptive planning. In *Proceedings of the international conference on automated planning and scheduling*, volume 30, 440–448.

Glorot, X.; Bordes, A.; and Bengio, Y. 2011. Deep sparse rectifier neural networks. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, 315–323.

Haarnoja, T.; Zhou, A.; Abbeel, P.; and Levine, S. 2018. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International conference on machine learning*, 1861–1870. Pmlr.

Hart, P. E.; Nilsson, N. J.; and Raphael, B. 1968. A formal basis for the heuristic determination of minimum cost paths. *IEEE transactions on Systems Science and Cybernetics*, 4(2): 100–107.

He, K.; Zhang, X.; Ren, S.; and Sun, J. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 770–778.

Ho, J.; Jain, A.; and Abbeel, P. 2020. Denoising diffusion probabilistic models. *Advances in neural information processing systems*, 33: 6840–6851.

Ioffe, S.; and Szegedy, C. 2015. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*.

Kim, B.; Kaelbling, L.; and Lozano-Pérez, T. 2018. Guiding search in continuous state-action spaces by learning an action sampler from off-target search experience. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32.

Kim, B.; Kaelbling, L. P.; and Lozano-Pérez, T. 2019. Adversarial actor-critic method for task and motion planning problems using planning experience. In *Proceedings of the AAAI conference on artificial intelligence*, volume 33, 8017–8024.

Kim, B.; Lee, K.; Lim, S.; Kaelbling, L.; and Lozano-Pérez, T. 2020. Monte carlo tree search in continuous spaces using voronoi optimistic optimization with regret bounds. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, 9916–9924.

Kingma, D. P.; and Ba, J. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Kingma, D. P.; and Welling, M. 2013. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*.

Lee, J.; Jeon, W.; Kim, G.-H.; and Kim, K.-E. 2020. Monte-carlo tree search in continuous action spaces with value gradients. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, 4561–4568.

Li, T.; Chen, R.; Mavrin, B.; Sturtevant, N. R.; Nadav, D.; and Felner, A. 2022. Optimal search with neural networks: Challenges and approaches. In *Proceedings of the International Symposium on Combinatorial Search*, volume 15, 109–117.

Mark, M. S.; Gao, T.; Sampaio, G. G.; Srirama, M. K.; Sharma, A.; Finn, C.; and Kumar, A. 2024. Policy agnostic rl: Offline rl and online rl fine-tuning of any class and backbone. *arXiv preprint arXiv:2412.06685*.

Mazoure, B.; Doan, T.; Durand, A.; Pineau, J.; and Hjelm, R. D. 2020. Leveraging exploration in off-policy algorithms via normalizing flows. In *Conference on Robot Learning*, 430–444. PMLR.

McAleer, S.; Agostinelli, F.; Shmakov, A.; and Baldi, P. 2019. Solving the Rubik's Cube with Approximate Policy Iteration. In *International Conference on Learning Representations*.

Mern, J.; Yildiz, A.; Sunberg, Z.; Mukerji, T.; and Kochenderfer, M. J. 2021. Bayesian optimized monte carlo planning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, 11880–11887.

Mnih, V.; Kavukcuoglu, K.; Silver, D.; Rusu, A. A.; Veness, J.; Bellemare, M. G.; Graves, A.; Riedmiller, M.; Fidjeland, A. K.; Ostrovski, G.; et al. 2015. Human-level control through deep reinforcement learning. *Nature*, 518(7540): 529–533.

Oh, J.; Guo, X.; Lee, H.; Lewis, R. L.; and Singh, S. 2015. Action-conditional video prediction using deep networks in atari games. *Advances in neural information processing systems*, 28.

Panta, R.; Tavakoli, M.; Geils, C.; Baldi, P.; and Agostinelli, F. 2024. Finding Reaction Mechanism Pathways with Deep Reinforcement Learning and Heuristic Search. In *ICAPS PRL Workshop*. ICAPS.

Pohl, I. 1970. Heuristic search viewed as path finding in a graph. *Artificial intelligence*, 1(3-4): 193–204.

Rezende, D.; and Mohamed, S. 2015. Variational inference with normalizing flows. In *International conference on machine learning*, 1530–1538. PMLR.

Rumelhart, D. E.; Hinton, G. E.; and Williams, R. J. 1986. Learning representations by back-propagating errors. *nature*, 323(6088): 533–536.

Schmidhuber, J. 2015. Deep learning in neural networks: An overview. *Neural networks*, 61: 85–117.

Sharon, G.; Stern, R.; Felner, A.; and Sturtevant, N. R. 2015. Conflict-based search for optimal multi-agent pathfinding. *Artificial intelligence*, 219: 40–66.

Silver, D.; Lever, G.; Heess, N.; Degris, T.; Wierstra, D.; and Riedmiller, M. 2014. Deterministic policy gradient algorithms. In *International conference on machine learning*, 387–395. Pmlr.

Sohn, K.; Lee, H.; and Yan, X. 2015. Learning structured output representation using deep conditional generative models. *Advances in neural information processing systems*, 28.

Standley, T. 2010. Finding optimal solutions to cooperative pathfinding problems. In *Proceedings of the AAAI conference on artificial intelligence*, volume 24, 173–178.

Sutton, R. S. 1991. Dyna, an integrated architecture for learning, planning, and reacting. *ACM Sigart Bulletin*, 2(4): 160–163.

Sutton, R. S.; and Barto, A. G. 2018. *Reinforcement learning: An introduction*. MIT press.

Tessler, C.; Tennenholtz, G.; and Mannor, S. 2019. Distributional policy optimization: An alternative approach for continuous control. *Advances in Neural Information Processing Systems*, 32.

Tian, S.; Nair, S.; Ebert, F.; Dasari, S.; Eysenbach, B.; Finn, C.; and Levine, S. 2021. Model-Based Visual Planning with Self-Supervised Functional Distances. In *International Conference on Learning Representations*.

Turner, I.; Agostinelli, F.; and Fu, P. 2025. Quantum Circuit Synthesis with Deep Reinforcement Learning and Heuristic Search. In *ICAPS PRL Workshop*. ICAPS.

Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A. N.; Kaiser, Ł.; and Polosukhin, I. 2017. Attention is all you need. *Advances in neural information processing systems*, 30.

Wang, Z.; Hunt, J. J.; and Zhou, M. 2023. Diffusion policies as an expressive policy class for offline reinforcement learning. *International Conference on Learning Representations*.

Ward, P. N.; Smofsky, A.; and Bose, A. J. 2019. Improving exploration in soft-actor-critic with normalizing flows policies. *arXiv preprint arXiv:1906.02771*.

Williams, R. J. 1992. Simple statistical gradient-following algorithms for connectionist reinforcement learning. In *Reinforcement Learning*, 5–32. Springer.

Yee, T.; Lisỳ, V.; Bowling, M. H.; and Kambhampati, S. 2016. Monte Carlo Tree Search in Continuous Action Spaces with Execution Uncertainty. In *IJCAI*, 690–697.

Yoshizumi, T.; Miura, T.; and Ishida, T. 2000. A* with Partial Expansion for Large Branching Factor Problems. In *AAAI/IAAI*, 923–929.

Zhang, Y.-H.; Zheng, P.-L.; Zhang, Y.; and Deng, D.-L. 2020. Topological Quantum Compiling with Reinforcement Learning. *Physical Review Letters*, 125(17): 170501.